Escola de Matemática Aplicada

Aula Prática 3 - Álgebra Linear Numérica

Aluno: *Gustavo Murilo Cavalcante Carvalho*
Programa: *Bacharelado em Matemática Aplicada*
Disciplina: *Álgebra Linear Numérica*
Professor: *Antonio Carlos Saraiva Branco*
Email: *gustavomurilo012@gmail.com*

Data: *2 de maio de 2024*

Sumário

Problema 1 - Método da Potência

Escreva uma função no MATLAB
`function [lambda,x1,k,n_erro] = Metodo_potencia(A,x0,epsilon,M)`
que implementa o Método da Potência para determinar o autovalor dominante λ_1 de A.

Implementação:

Versão 1 - Método da Potência

```
% ////////////////////////////////////////////////////////////////////
% Variáveis de entrada:
% A: matriz real n x n, diagonalizável, com autovalor dominante
   (lambda);
% x0: vetor, não nulo, a ser utilizado como aproximação inicial do
   autovetor dominante;
% epsilon: precisão a ser usada no critério de parada;
% M: número máximo de iterações.

% Variáveis de saída:
% lambda: autovalor dominante de A;
% x1: autovetor unitário (norma infinito) correspondente a lambda;
% k: número de iterações necessárias para a convergência;
% n_erro: norma infinito do erro.

% Critério de parada: sendo erro=x1 - x0 (diferença entre dois
% iterados consecutivos), parar quando n_erro < épsilon ou k>M.
% ////////////////////////////////////////////////////////////////////
function [lambda, x1, k, n_erro] = Metodo_potencia_v1(A, x0, epsilon, M)

  % Definição de variáveis para os critérios de parada
  k = 0;
  n_erro = epsilon + 1; % Valor para entrar no loop

  % Primeira iteração do método
  x0 = x0 / norm(x0, inf);
  x1 = A*x0;

  while (k < M && n_erro > epsilon)
```

```
28      lambda = max(x1, [], 'ComparisonMethod','abs'); % Coordenada de maior
        ↪  modulo
29
30      % Normalização da (k)-ésima aproximação do autovetor
31      x1 = x1 / lambda;
32
33      % Cálculo da diferença entre as aproximações
34      n_erro = norm(x1 - x0, inf);
35
36      x0 = x1; % atualiza (k)-ésima aproximação do autovetor
37
38      % Cálculo da (k+1)-ésima aproximação do autovetor
39      x1 = A * x0;
40
41      % Atualização das condições de parada
42      k = k + 1;
43    end
44
45    if (k >= M)
46      fprintf('O método atingiu o limite de %d iterações. \n',M);
47    end
48
49  end % Finaliza a função
```

Versão 2 - Método da Potência

```
1   % ////////////////////////////////////////////////////////////////////
2   % Variáveis de entrada:
3   % A: matriz real n x n, diagonalizável, com autovalor dominante
    ↪  (lambda);
4   % x0: vetor, não nulo, a ser utilizado como aproximação inicial do
    ↪  autovetor dominante;
5   % epsilon: precisão a ser usada no critério de parada;
6   % M: número máximo de iterações.
7
8   % Variáveis de saída:
9   % lambda: autovalor dominante de A;
10  % x1: autovetor unitário (norma 2) correspondente a lambda;
```

```
% k: número de iterações necessárias para a convergência;
% n_erro: norma euclidiana do erro.

% Critério de parada: sendo erro=x1 { x0 (diferença entre dois
% iterados consecutivos), parar quando n_erro < épsilon ou k>M.
% ////////////////////////////////////////////////////////////////
function [lambda, x1, k, n_erro] = Metodo_potencia_v2(A, x0, epsilon, M)

  % Definição de variáveis para os critérios de parada
  k = 0;
  n_erro = epsilon + 1; % Valor para entrar no loop

  % Primeira iteração do método
  x0 = x0 / norm(x0);
  x1 = A*x0;

  while (k < M && n_erro > epsilon)
    % Aproximação do autovalor dominante
    lambda = x1' * x0; % (Quociente de Rayleigh, dado que x0 é unitario)

    % Orienta x1 no sentido de x0
    if lambda < 0
      x1 = -x1;
    end

    % Normalização da (k)-ésima aproximação do autovetor
    x1 = x1 / norm(x1);

    % Cálculo da diferença entre as aproximações
    n_erro = norm(x1 - x0);

    x0 = x1; % atualiza (k)-ésima aproximação do autovetor

    % Cálculo da (k+1)-ésima aproximação do autovetor
    x1 = A*x0;

    k = k + 1; % Atualização de condição de parada
  end

```

```
50    if (k >= M)
51      fprintf('O método atingiu o limite de %d iterações. \n',M);
52    end
53
54  end % Finaliza a função
```

Problema 2 - Método da Potência Deslocada com Iteração Inversa

MÉTODO DA POTÊNCIA DESLOCADA com ITERAÇÃO INVERSA Escreva uma função Scilab function [lambda1,x1,k,n˙erro] = Potencia˙deslocada˙inversa (A,x0,epsilon,alfa,M) que implementa o Método da Potência Deslocada com Iteração Inversa para determinar o autovalor de A mais próximo de "alfa".

Implementação:

```
% ////////////////////////////////////////////////////////////////////
% Variáveis de entrada:
% A: matriz real n x n, diagonalizável;
% x0: vetor, não nulo, a ser utilizado como aproximação inicial do
↪  autovetor dominante.
% epsilon: precisão a ser usada no critério de parada.
% alfa: valor do qual se deseja achar o autovalor de A mais próximo;
% M: número máximo de iterações.

% Variáveis de saída:
% lambda1: autovalor de A mais próximo de alfa;
% x1: autovetor unitário (norma_2) correspondente a lambda;
% k: número de iterações necessárias para a convergência
% n_erro: norma_2 do erro

% Critério de parada: sendo erro = x1 - x0 (diferença entre dois
% iterados consecutivos),
% parar quando a n_erro < épsilon ou k>M.
% ////////////////////////////////////////////////////////////////////
function [lambda1,x1,k,n_erro] = Potencia_deslocada_inversa(A, x0,
↪  epsilon, alfa, M)

  % Decomposiçao L*U de P*(A - alfa*I)
  n = size(A,1);
  B = A - alfa*eye(n);
  [~, C, P] = Gaussian_Elimination_4(B, x0);

  % Verificação da invertibilidade da matriz deslocada
```

```
27    if min(abs(diag(C))) == 0
28      fprintf('\n(A - %d*I) nao é invertível, portanto %d é autovalor de
        ↪  A.\n\n',alfa,alfa);
29      error('O alfa é autovalor de A, o que acarreta divisões por 0.');
30    end
31
32    % Separação da fatoração L*U compactada em C
33    L = tril(C, -1) + eye(n);
34    U = triu(C);
35
36    % Definição de variáveis para os critérios de parada
37    k = 0;
38    n_erro = epsilon + 1; % Para entrar no loop
39
40    % Normaliza x0
41    x0 = x0 / norm(x0);
42
43    % Iterações até a convergência ou M iterações
44    while k <= M && n_erro > epsilon
45      % Dado (A { alfa*I)*x1 = x0 e P*(A { alfa*I) = L*U,
46      % Resolve L*U*x1 = P*x0
47      x1 = Resolve_LU(L, U, P*x0);
48
49      % Normalizaação do (k)-ésimo autovetor dominante
50      x1 = x1 / norm(x1);
51
52      % Calcula lambda usando o quociente de Rayleigh
53      lambda1 = x1' * A * x1;
54
55      % Mantém x1 e x2 no mesmo sentido
56      if x1' * x0 < 0
57        x1 = -x1;
58      end
59
60      % Calcula o erro
61      n_erro = norm(x1 - x0);
62
63      % Atualiza x0 para a próxima iteração
64      x0 = x1;
```

```
    k = k + 1;
  end

  if k >= M
    fprintf('O metodo atingiu o limite de %d iteraçoes. \n',M);
  end

end % Finaliza a função
```

Funções auxiliares:

```
function[x] = Resolve_LU(L,U,b)
  n = size(L,1);
  y = zeros(n,1);
  x = zeros(n, 1);

  % Resolve o sistema triangular inferior
  % L*y = b, sendo a diagonal de L composta por 1's.
  y(1) = b(1);
  for i = 2 : n
    y(i) = b(i) - (L(i, 1 : i-1) * y(1 : i-1));
  end

  % Resolve o sistema triangular superior
  % U*x = y
  x(n) = y(n) / U(n,n);
  for j = n-1 : -1 : 1
    x(j) = (y(j) - U(j, j+1 : n) * x(j+1 : n)) / U(j,j);
  end

end % Finaliza a função
```

```
function [x, C, P] = Gaussian_Elimination_4(A, b)
  C = [A, b];
  [n] = size(C, 1);
  P = eye(n);

```

```
6   for j = 1:(n-1)
7     % Obtem o valor do maior pivo e a distancia a coluna j
8     [maior_pivo, dist] = max(abs(C(j:n,j)));
9
10    % Caso onde a matriz A não é singular. Não existe decomposição LU.
11    if maior_pivo == 0
12      error("O sistema é indeterminado");
13    end
14
15    % Troca as linhas caso o maior pivo esteja abaixo da linha j
16    if dist ~= 1
17      k = j + dist - 1;
18
19      P([j k], :) = P([k j], :);
20      C([j k], :) = C([k j], :);
21    end
22
23    % Processo de decomposição LU de A.
24    for i = (j+1):n
25      % O elemento C(i,j) é o elemento na posição (i,j) de L na
        ↪  decomposição LU de A.
26      C(i, j) = C(i, j) / C(j, j);
27      % Linha i <- Linha i - C(i,j)*Linha j.
28      % Somente os elementos da diagonal ou acima dela são computados
29      % (aqueles que compõem a matriz U).
30      C(i, j+1:n+1) = C(i, j+1:n+1) - C(i, j) * C(j, j+1:n+1);
31    end
32  end
33
34  x = zeros(n, 1);
35
36  % Calcula x, sendo Ux=C(1:n,n+1).
37  x(n) = C(n, n+1) / C(n, n);
38  for i = n-1:-1:1
39    x(i) = (C(i, n+1) - C(i, i+1:n) * x(i+1:n)) / C(i, i);
40  end
41  C = C(1:n, 1:n);
42
43 end % Fim da função
```

Problema 3

Teste suas duas primeiras funções para várias matrizes A, com ordens diferentes e também variando as demais variáveis de entrada de cada função. Use matrizes com autovalores reais (por exemplo, matrizes simétricas ou matrizes das quais você saiba os autovalores). Teste a mesma matriz com os dois primeiros algoritmos, comparando os números de iterações necessárias para convergência e os tempos de execução. Teste com uma matriz em que o autovalor dominante é negativo. Alguma coisa deu errada? Se for o caso, corrija o algoritmo (e a função) correspondente.

Solução:

Nada mais justo que começar os testes com as matrizes e vetores do capítulo 4.5 (Métodos Iterativos para o Cálculo de Autovalores) do Poole. Sejam as matrizes:

$$A_1 = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} \qquad x_1^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad A_2 = \begin{bmatrix} 0 & 5 & -6 \\ -4 & 12 & -12 \\ -2 & -2 & 10 \end{bmatrix} \qquad x_2^{(0)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Notação: $x^{(k)}$ indica a k-ésima aproximação de x (autovetor dominante de A) obtida pelo método iterativo.

Pelo método da potência, temos que:

$$x_i^{(k)} = (A_i)^k \, x_i^{(k-1)}$$

Considere o i-ésimo exemplo como a aplicação do método da potência para a matriz A_i e o palpite de autovetor dominante $x_i^{(0)}$.

Ao executar uma das funções para o método da potência repetidas vezes sob determinadas condições teremos sempre as mesmas saídas, a única coisa que pode variar é o tempo de execução, por isso foi criada uma função ("Comparar_v1_v2", que está no fim do relatório) para verificar isso, executando as funções várias vezes e plotando o seu tempo de execução em cada teste.

Verificação das funções

Exemplo 1:

```
>> [lambda, x1, k, n_erro] = Metodo_potencia_v1(A_1, x0_1, 1e-6, 100)

    lambda =       x1 =           k =            n_erro =
        2              2             22             7.1526e-07
                       2

>> [lambda, x1, k, n_erro] = Metodo_potencia_v2(A_1, x0_1, 1e-6, 100)
```

```
    lambda =       x1 =           k =            n_erro =
        2              1.4142         22             5.3644e-07
                       1.4142
```

Essa matriz possus os autovalores 2 e -1, então ambas as funções são bem sucedidas em achar o autovalor dominante e fazem isso com o mesmo número de iterações. A segunda versão consegue um erro menor, mas isso não é muito significativo pois ambas já estão abaixo do erro máximo estipulado.

Exemplo 2:

```
>> [lambda, x1, k, n_erro] = Metodo_potencia_v1(A_2, x0_2, 1e-6, 100)
    lambda =       x1 =           k =            n_erro =
        16              8             11             7.1689e-07
                       16
                       -8

>> [lambda, x1, k, n_erro] = Metodo_potencia_v2(A_2, x0_2, 1e-6, 100)

    lambda =       x1 =           k =            n_erro =
        16           -6.532           11             5.3446e-07
                   -13.064
                     6.532
```

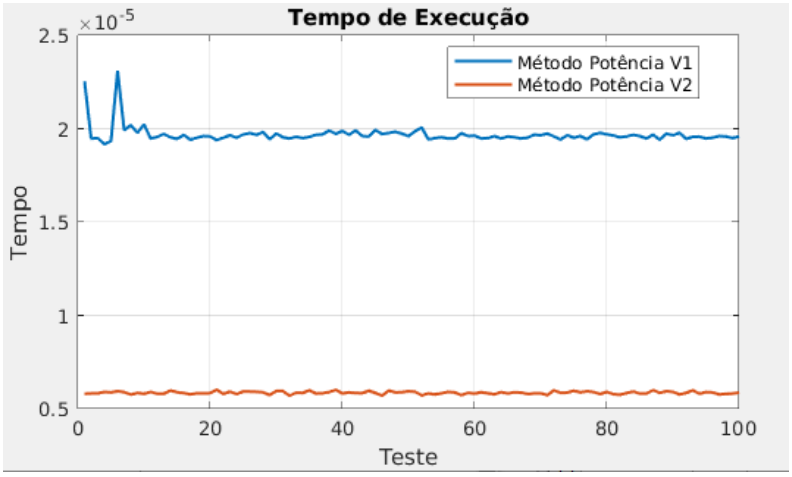
Nesse segundo exemplo, a matriz tem os autovalor 16, 4 e 2. Então, assim como no exemplo anterior, ambas conseguem o resultado esperado com o mesmo número de iterações, sendo que a segunda versão possui um erro menor.

Comparação do tempo de execução para matrizes pequenas

Considerando os dois exemplos anteriores, executo uma função para capturar o tempo de execução de cada versão n vezes, plotar um gráfico e calcular o tempo médio de cada versão. A função em questão pode ser vista no fim do relatório na seção "Funções/Scripts Extra".

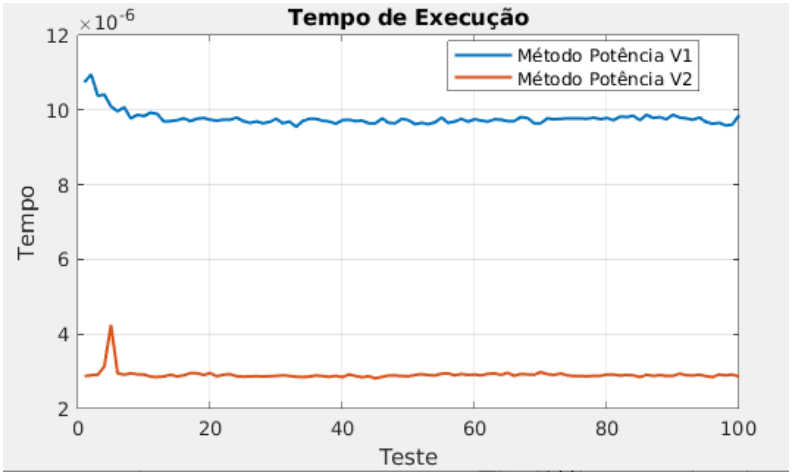
Dentro dessa função, o tempo é medido pela função timeit() nativa do MATLAB, ela é o meio mais adequado para esse propósito, aliás ela é mais precisa que as funços tic() e toc() usadas na aula prática anterior.

Constam abaixo, respectivamente, as medidas relativas aos exemplos 1 e 2, considerando $n = 100$ (número de repetições), o número de testes ideal obtido empiricamente.



Tempo médio da versão 1: 19.684 microssegundos

Tempo médio da versão 2: 5.857 microssegundos



Tempo médio da versão 1: 9.954 microssegundos

Tempo médio da versão 2: 2.932 microssegundos

Perceba que nesses exemplos, a segunda versão se mostra um pouco mais rápida, essa diferença está na escala de 10^{-6} segundos, o que só vai ser relevante se elas precisarem ser usadas muitas vezes.

Matriz com autovalor negativo

A matriz A_2 tem autovalores 16, 4 e 2. Logo, a matriz $(-A_2)$ tem os autovalores -16, -4 e -2, todos negativos. Aplicando as funções a essa matriz, obtemos um resultado similar ao da anterior, mas o autovetor e autovalor tem o sinal trocado. Portanto, o método é bem sucedido em encontra o maior autovalor em módulo.

```
>> [lambda, x1, k, n_erro] = Metodo_potencia_v1(A_2, x0_2, 1e-6, 100)

    lambda =       x1 =           k =            n_erro =
        -16            -8             11             7.1689e-07
                      -16
                        8

>> [lambda, x1, k, n_erro] = Metodo_potencia_v2(A_2, x0_2, 1e-6, 100)

    lambda =       x1 =           k =            n_erro =
```

```
-16          6.532          11          5.3446e-07
             13.064
             -6.532
```

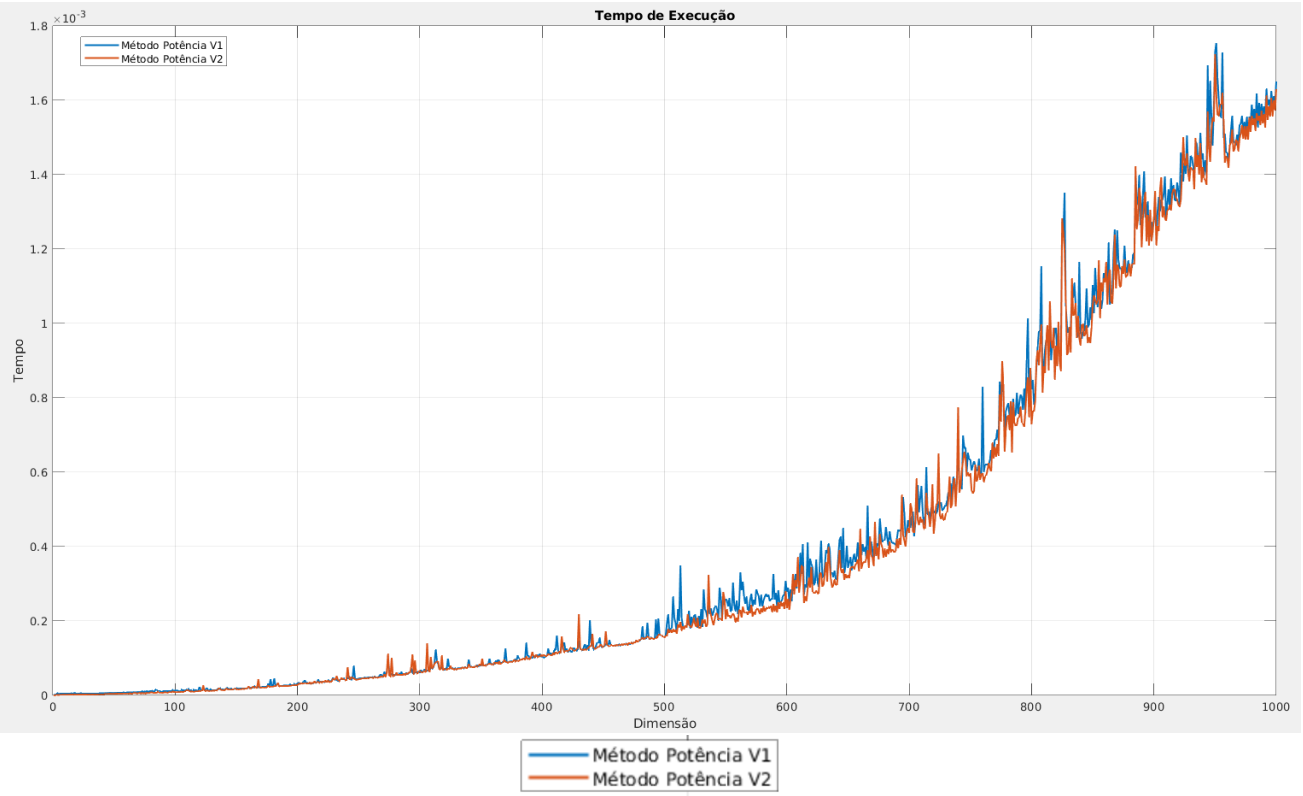
Comparação do tempo de execução para matrizes de diferentes dimensões

Para os exemplos anteriores, a versão 2 se mostrou ligeiramente mais rápida que a versão 1. Verificar o tempo de execução para apenas duas matriz não nos diz muito sobre o desempenho geral.

Com isso mente, irei verificar o tempo de execução para matrizes de dimensões de 1 a 1000, todas simétricas e compostas por inteiros positivos distribuidos aleatoriamente.

A função utilizada para isso é a "Comparar_tempo_dimensoes", ela calcula o tempo médio das funções para cada matriz 10 vezes, então essa médida é plotada. Além de plotar, ela imprime a média da diferença das versões.

Pelo teste feito, verifiquei que a versão 2 é em media 13.659 microsegundos mais rápida que a versão 1. Como dito anteriormente, isso pode ou não ser relevante, depende do propósito da aplicação. Para o uso doméstico, essa diferença é inperceptível.



Problema 4

Construa uma matriz simétrica e use os Discos de Gerschgorin para estimar os autovalores. Use essas estimativas e o Método da Potência Deslocada com Iteração Inversa para calcular os autovalores. Alguma coisa deu errada? Comente!

Estamos considerandos matrizes com valores reais. Assim, os centros dos discos irão coincidir com a reta real, parte do plano de Argand-Gauss onde $\beta = 0$, ou seja, $z = \alpha + \beta i = a$ com $z \in \mathbb{C}$ e $\alpha, \beta \in \mathbb{R}$.

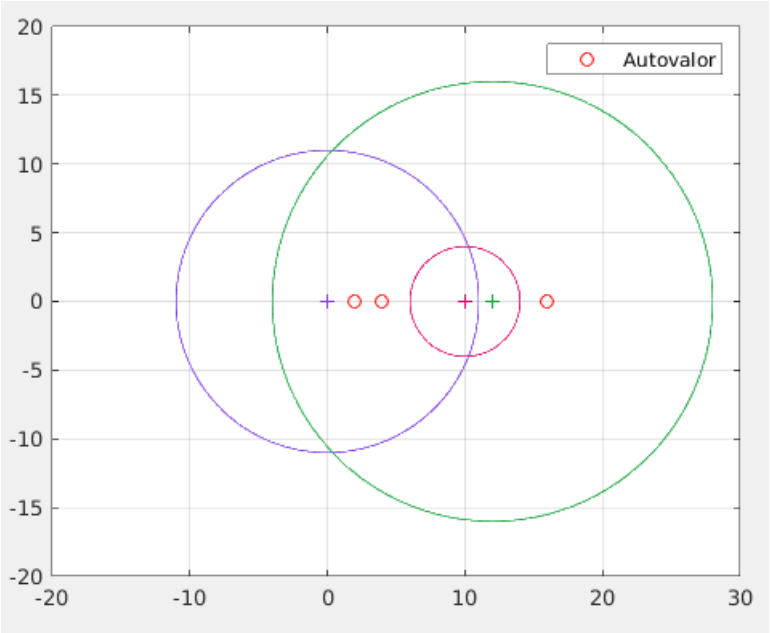
Criei uma função ("Discos_Gershgorin") que plota os Discos de Gerschgorin e os autovalores de uma matriz. Para ser coerente com o objetivo da questão, na qual não sabemos os autovalores, levarei em conta a seguinte premissa: ao usar os discos, pegarei como potenciais α os centros c dos dicos de raio r e os seus extremos na reta real $c \pm r$.

$$A_2 = \begin{bmatrix} 0 & 5 & -6 \\ -4 & 12 & -12 \\ -2 & -2 & 10 \end{bmatrix}$$

Considere novamente a matriz A_2 usada no problema anterior. Aplicamos essa nva função a ela, obtendo o seguinte retorno:

```
>> alfas = Discos_Gershgorin(A_2, "lin")
```

```
alfas =
     0
    12
    10
   -11
    -4
     6
    14
     0
     0
```



para automatizar o teste com esses possíveis α, criei a seguinte função:

```
function [resultados] = Testar_alfas(A, x0, alfas)
  % Definição de variáveis
  n = size(A,1);
  t = 3*n;
  lambdas = zeros(1, t);
  iteracoes = zeros(1, t);

  % TestesAutovalores obtidos
  for i = 1 : t
    [lambdas(i), ~, iteracoes(i), ~] = Potencia_deslocada_inversa(A, x0,
    ↪  1e-12, alfas(i), 200);
  end

  resultados = [lambdas; iteracoes];
  rotulos = ["Lambda: "; "k: "];

  resultados = [rotulos, resultados];
end
```

Nesse teste, o erro máximo é na escala e 10^{-12} e o número máximo de itrerações é 200.

Ao executar a função obtemos o seguinte resultado, onde o indicie i corresponde ao $(i-1)$-ésimo elemento da variável "alfas".

	1	2	3	4	5	6	7	8	9	10
1	Lambda:	2	16	10	2	2	4	16	2	2
2	k:	37	42	201	166	85	40	19	37	37

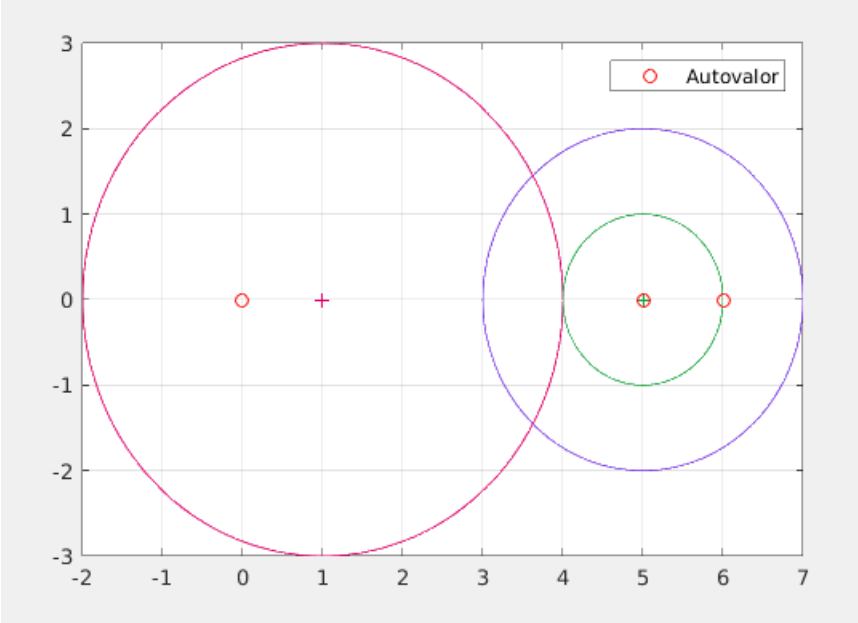
Relembrando, os autovalores de A_2 são 2, 4 e 16.

Logo, usando esses pontos selecionados, todos os autovalores foram encontrados pela função "Potencia_deslocada_inversa" para essa matriz.

Note que na coluna 4, temos lambda = 10, que não é um autovalor de A_2. Isso acontece pois 10 está equidistante de dois autovalores, sendo eles 4 e 16. Nesse caso, o método não consegue progredir e itera o máximo de vezes o possível, mantendo lambda = alfa, do começo ao fim.

Essa ideia funcionaria também para o seguinte exemplo:

$$A_3 = \begin{bmatrix} 5 & 0 & 2 \\ 0 & 5 & 1 \\ 2 & 1 & 1 \end{bmatrix}$$



Problema 5

Faça outros testes que achar convenientes ou interessantes!!! ☺

Solução:

Não pensei em nada além do que foi abordado nos problemas anterirores :D

Funções/Scripts Extra

Teste de velocidade das versões 1 e 2

```
function Comparar_v1_v2(repeticoes, A, x0, E, M)

tempos_1 = zeros(repeticoes, 1);
tempos_2 = zeros(repeticoes, 1);

for i = 1 : repeticoes
  % Tempo da primeira versao da função
  v1 = @() Metodo_potencia_v1(A, x0, E, M);
  tempos_1(i) = timeit(v1);

  % Tempo da segunda versao da função
  v2 = @() Metodo_potencia_v2(A, x0, E, M);
  tempos_2(i) = timeit(v2);
end

% Plotando os resultados dos tempos de execução
plot(tempos_1, 'DisplayName', 'Método Potência V1', 'LineWidth', 1.5);
hold on;
plot(tempos_2, 'DisplayName', 'Método Potência V2', 'LineWidth', 1.5);
hold off;
title('Tempo de Execução');
xlabel('Teste');
ylabel('Tempo');
legend('Location', 'best');
grid on;

% Calcula a média dos tempos de execução
media_1 = mean(tempos_1);
media_2 = mean(tempos_2);

% Exibir média dos tempos de execução
fprintf('Tempo médio da versão 1: %.3f microsegundos\n', media_1 * 1e6);
fprintf('Tempo médio da versão 2: %.3f microsegundos\n', media_2 * 1e6);

```

```
end
```

Teste de velocidade das matrizes de dimensão de 1 a 1000

```
% E: tamanho do erro aceitavel
% M: numero maximo de iteraçoes do metodo
function Comparar_tempo_dimensoes(E, M)

% Repetições e pré-alocação de variáveis
repeticoes = 10;
n = 1000;

media_1 = zeros(200, 1);
media_2 = zeros(200, 1);

for i = 1 : n
    % Cria uma matriz quadrada com valores de 1 a 3
    A = randi(3, i, i);
    A = A' * A; % Cria matriz simétrica

    % Cria um palpite do autovetor dominante
    x0 = ones(i, 1);

    tempos_1 = zeros(repeticoes, 1);
    tempos_2 = zeros(repeticoes, 1);

    for j = 1 : repeticoes
        % Tempo da primeira versao da função
        v1 = @() Metodo_potencia_v1(A, x0, E, M);
        tempos_1(j) = timeit(v1);

        % Tempo da segunda versao da função
        v2 = @() Metodo_potencia_v2(A, x0, E, M);
        tempos_2(j) = timeit(v2);
    end

    % Calcula a média dos tempos
    media_1(i) = mean(tempos_1);
```

```
35      media_2(i) = mean(tempos_2);
36  end
37
38  % Plotando os resultados dos tempos de execução
39  figure;
40  plot(1:n, media_1, 'DisplayName', 'Método Potência V1', 'LineWidth',
    ↪  1.5);
41  hold on;
42  plot(1:n, media_2, 'DisplayName', 'Método Potência V2', 'LineWidth',
    ↪  1.5);
43  hold off;
44  title('Tempo de Execução');
45  xlabel('Dimensão');
46  ylabel('Tempo');
47  legend('Location', 'best');
48  grid on;
49
50
51  % Calcula a média dos tempos de execução
52  dif = mean(media_1 - media_2);
53
54  % Exibir média dos tempos de execução
55  fprintf('A versão 2 é em media %.3f microsegundos\n mais rápida que a
    ↪  versão 1.', dif * 1e6);
56
57  end
```

Discos de Gershgorin

```
1  % ////////////////////////////////////////////////////////////////
2  % Variáveis de entrada:
3  % A: matriz nxn;
4  % s: sentido do calculo do raio ("col", "lin").
5  % ////////////////////////////////////////////////////////////////
6  function [alfas] = Discos_Gershgorin(A, s)
7    % Definiçao de variaveis
8    n = size(A, 1);
9    d = diag(A);
```

```
10    M = abs(A);
11    centros = [real(d), imag(d)];
12    raios = zeros(n, 1);
13    alfas = zeros(3*n, 1);
14
15    % Localizaçao dos autovetores no plano de Argand-Gauss
16    eigval = [real(eig(A)), imag(eig(A))];
17
18    % Caclculo dos raios dos discos de Gershgorin
19    for i = 1 : n
20      if (s == "col")
21        % Soma na coluna
22        raios(i) = sum(M(1:i-1, i)) + sum(M(i+1:n, i));
23      else
24        % Soma na linha
25        raios(i) = sum(M(i, 1:i-1)) + sum(M(i, i+1:n));
26      end
27
28      %Define os pontos de interesse
29      alfas(i) = centros(i, 1);
30      alfas(n+i) = centros(i, 1) - raios(i);
31      alfas(n+i+1) = centros(i, 1) + raios(i);
32    end
33
34    D = [centros, raios];
35
36    Plotar_Discos(D,eigval); % Plota os discos de Gershgorin
37  end
38
39  function Plotar_Discos(D,eigval)
40    n = size(D,1);
41
42    % Plotagem dos Circulos
43    for i = 1 : n
44      center = Desenhar_Circulo(D(i,:));
45    end
46
47    % Plotagem de marcadores para so autovalores de A
48    for i = 1 : n
```

```
49      eigloc = plot(eigval(i,1), eigval(i,2), 'ro');
50    end
51
52    % Opçoes de plotagem
53    axis normal;
54    grid on;
55    hold off;
56
57    mycolors =
      ↪  ["#8040E6";"#8040E6";"#1AA640";"#1AA640";"#DF0069";"#DF0069";];
58    colororder(mycolors)
59
60    legend(eigloc, 'Autovalor');
61  end
62
63  function [center] = Desenhar_Circulo(v)
64    % Parametros do disco
65    xCentro = v(1);
66    yCentro = v(2);
67    raio = v(3);
68
69    % Conjunto de pontos da parametrizaçao
70    theta = 0 : 0.01 : 2*pi;
71    x = raio * cos(theta) + xCentro;
72    y = raio * sin(theta) + yCentro;
73
74    % Plotagem do circulo
75    plot(x, y);
76    hold on;
77
78    % Plotagem de marcador para o centro
79    center = plot(xCentro, yCentro, '+');
80    hold on;
81  end
```